**Comanda SELECT**

Interogarea datelor stocate in baza de date este considerata cea mai importanta facilitate a unui SGBD. In SQL ea se realizeaza prin intermediul comenzii SELECT. Comanda SELECT este folosita doar pentru interogarea datelor, ea neputandu-le modifica. Asa cum am vazut mai inainte, comanda SELECT implementeaza toti operatorii algebrei relationale. O instructiune SELECT cuprinde in mod obligatoriu cuvantul cheie FROM. Cu alte cuvinte, sintaxa minimala pentru comanda SELECT este:

http://ib.adnxs.com/getuid?http%3a%2f%2fdis.criteo.com%2frex%2fmatch.aspx%3fc%3d11%26uid%3d%24UID

     SELECT atribute

     FROM obiect

Dupa cuvantul cheie SELECT se specifica lista atributelor ce urmeaza a fi returnate ca rezultat al interogarii, iar dupa cuvantul FROM se precizeaza obiectele (tabele, vederi, sinonime) din care se vor selecta aceste atribute.

**5.1.1. Atributele comenzii SELECT**

In lista de atribute pot aparea:

        toate coloanele din tabel sau vedere (in ordinea in care au fost definite in comanda CREATE TABLE/CREATE VIEW) prin utilizarea semnului \*:

SQL> SELECT \*

     FROM profesor;

COD  NUME    PRENUME   DATA\_NAST  GRAD   SEF  SALARIU PRIMA  COD\_CATEDRA

--- --------- --------- ---------  ----   ---  ------- -----  -----------

100 GHEORGHIU STEFAN    ll-AUG-46  PROF         3000   3500      10

101 MARIN     VLAD      19-APR-45  PROF   100   2500              20

102 GEORGESCU CRISTIANA 30-OCT-51  CONF   100   2800     200      30

103 IONESCU   VERONICA             ASIST  102   1500              10

104 ALBU      GHEORGHE             LECT   100   2200    2500      20

105 VOINEA    MIRCEA    15-NOV-65  ASIST  100   1200     150      10

106 STANESCU  MARIA     05-DEC-69  ASIST  103   1200     600      20

        numele coloanelor separate prin virgula. Acestea vor aparea in rezultatul interogarii in ordinea in care sunt specificate:

SQL> SELECT nume, prenume, salariu

     FROM profesor;

NUME          PRENUME      SALARIU

---------     -------       -------

GHEORGHIU     STEFAN        3000

MARIN         VLAD          2500

GEORGESCU     CRISTIANA     2800

IONESCU       VERONICA      1500

ALBU      GHEORGHE       2200

VOINEA    MIRCEA         1200

STANESCU MARIA         1200

        atribute rezultate din evaluarea unor expresii. Aceste expresii pot contine nume de coloane, constante, operatori sau functii.

**5.1.2. Operatori aritmetici**

Operatorii aritmetici pot fi folositi pentru a crea expresii avand tipul de date numeric sau date calendaristice. Operatorii aritmetici sunt + (adunare), - (scadere ), \* (inmultire), / (impartire). Ordinea de precedenta a operatorilor poate fi schimbata cu ajutorul parantezelor.

De exemplu daca in tabela profesor ne intereseaza sa calculam impozitul aferent salariilor, stiind ca acesta este de 38%, putem scrie urmatoarea interogare:

SQL> SELECT nume, salariu, salariu\*0.38

     FROM profesor;

NUME      SALARIU   SALARIU\*0.38

----      -------   ------------

GHEORGHIU   3000        1140

MARIN       2500         950

GEORGESCU   2800        1064

IONESCU     1500         570

ALBU        2200         836

VOINEA      1200         456

STANESCU   1200         456

**5.1.3. Aliasuri de coloane**

Pentru exemplul anterior observam ca in momentul afisarii rezultatelor, se mai utilizeaza numele coloanelor ca antet. Cand acest lucru poate face dificila intelegerea rezultatelor, se poate schimba antetul prin atribuirea altor nume coloanelor (numite 'alias'-uri ale coloanei). Acest lucru se realizeaza specificand alias-ul dupa numele coloanei. In cazul in care alias-ul contine spatii sau caractere speciale cum ar fi + sau -, acesta se va specifica intre ghilimele.

In exemplul de mai jos alias-ul 'DATA  NASTERE' contine spatii deci este specificat intre ghilimele in timp ce alias-ul impozit nu con 636i89g tine spatii sau caractere speciale deci nu trebuie specificai obligatoriu intre ghilimele.

SQL> SELECT nume, data\_nast 'DATA NASTERE', salariu,      salariu\*0.38 impozit

     FROM profesor;

NUME           DATA NASTERE  SALARIU        IMPOZIT

----           ------------  -------        -------

GHEORGHIU     ll-AUG-46     3000          1140

MARIN         19-APR-45     2500           950

GEORGESCU     30-OCT-51     2800          1064

IONESCU                      1500           570

ALBU                         2200           836

VOINEA         15-NOV-65      1200           456

STANESCU      05-DEC-69     1200            456

**5.1.4. Operatorul de concatenare**

Operatorul de concatenare, notat | |, permite legarea coloanelor cu alte coloane, expresii aritmetice sau valori constante pentru a crea o expresie de tip sir de caractere. De exemplu, pentru a combina codul, numele si prenumele unui profesor, separate printr-un spatiu, se foloseste urmatoarea interogare:

     SQL> SELECT cod || ' ' || nume || ' ' || prenume detalii      FROM profesor;

DETALII

-------

100 GHEORGHIU STEFAN

101 MARIN VLAD

102 GEORGESCU CRISTIANA

103 IONESCU VERONlCA

104 ALBU GHEORGHE

105 VOINEA MIRCEA

106 STANESCU MARIA

**5.1.5. Convertirea valorilor Null cu ajutorul functiei NVL**

Daca la o inregistrare pentru o anumita coloana valoarea este necunoscuta sau neaplicabila, atunci aceasta este Null. Aceasta valoare nu trebuie confundata cu zero sau sirul de caractere format dintr-un spatiu. Asa cum am vazut in exemplele de pana acum, daca o anumita valoare este Null, nu se va afisa nimic. Pentru expresiile aritmetice, daca una dintre valorile componente este Null, atunci si rezultatul expresiei este Null. De exemplu, pentru a calcula salariul total, ce reprezinta suma dintre coloanele salariu si prima putem folosi interogarea:

SQL> SELECT nume, salariu, prima,salariu+prima 'SALARIU TOTAL'     FROM profesor;

NUME      SALARIU   PRIMA  SALARIU TOTAL

----      -------   -----  -------------

GHEORGHIU 3000    3500      6500

MARIN     2500

GEORGESCU 2800     200       3000

IONESCU   1500

ALBU      2200      2500      4700

VOINEA    1200       150      1350

STANESCU 1200       600      1800

Observam ca pentru acele inregistrari care au avut valoarea Null in campul "prima" expresia ce calculeaza "salariul total" returneaza tot valoarea Null.

Pentru a obtine un rezultat diferit de Null, valorile Null trebuiesc convertite intr-un numar (in cazul de fata 0) inainte de a aplica operatorul aritmetic. Aceasta convertire se poate realiza prin intermediul functiei NVL. Functia NVL are doua argumente. Daca valoarea primului argument nu este Null, atunci NVL intoarce aceasta valoare; altfel, ea intoarce valoarea celui de-al doilea argument. Cele doua argumente pot avea orice tip de date. Daca tipurile de date ale celor doua argumente difera, Oracle incerca sa converteasca, al doilea argument la tipul de date al primului. De exemplu, pentru a putea calcula salariul total al tuturor cadrelor didactice, trebuie sa convertim valoarea Null din coloana "prima" a tabelei "professor" in valoarea 0 folosind NVL (prima, 0):

SQL> SELECT nume, salariu, prima, salariu+NVL(prima,0)'SALARIU TOTAL' FROM profesor;

NUME      SALARIU   PRIMA     SALARIU TOTAL

----      -------   -----     -------------

GHEORGHIU 3000     3500      6500

MARIN     2500               2500

GEORGESCU 2800      200      3000

IONESCU   1500               1500

ALBU      2200      2500      4700

VOINEA    1200       150      1350

STANESCU 1200       600      1800

**5.1.6. Prevenirea selectarii inregistrarilor duplicate**

O comanda SELECT care nu cuprinde cuvantul cheie DISTINCT va afisa toate inregistrarile care rezulta din interogare, indiferent daca unele dintre ele sunt identice. De exemplu, interogarea de mai jos va returna urmatoarele rezultate:

SQL> SELECT grad

     FROM profesor;

GRAD

----

PROF

PROF

CONF

ASIST

LECT

ASIST

ASIST

In cazul folosirii cuvantului cheie DISTINCT inregisrarile duplicat sunt eliminate, afisandu-se numai prima aparitie a valorilor campurilor specificate in lista de atribute. De exemplu:

SQL> SELECT DISTINCT grad

     FROM profesor;

GRAD

----

ASIST

CONF

LECT

PROF

Daca lista de atribute contine mai multe coloane, operatorul DISTINCT va afecta toate coloanele selectate. Urmatorul exemplu va afisa toate combinatiile de valori care sunt diferite pentru coloanele grad si cod\_catedra.

SQL> SELECT DISTINCT grad, cod\_catedra

     FROM profesor;

GRAD COD\_CATEDRA

---- -----------

ASIST     10

ASIST     20

CONF      30

LECT      20

PROF      10

PROF      20

**5.1.7. Clauza ORDER BY**

In mod normal, in urma interogarii inregistrarile rezultate apar in aceeasi ordine in care au fost introduse in baza de date. Pentru a modifica ordinea de afisare se utilizeaza clauza ORDER BY, care sorteaza inregistrarile dupa valorile din una sau mai multe coloane. Aceasta clauza este urmata de numele coloanelor dupa care se va face sortarea. De asemenea, este posibil sa se identifice coloana dintr-o clauza ORDER BY folosind in locul numelui coloanei un numar ordinal ce reprezinta pozitia coloanei in rezultat (de la stanga la dreapta). Aceasta facilitate face posibila ordonarea rezultatului interogarii in functie de un atribut al clauzei SELECT care poate fi o expresie complexa, fara a mai rescrie acea expresie.

Nu exista nici o limita a numarului de coloane in functie de care se poate face sortarea. Nu este obligatoriu ca ordinea de sortare sa se faca in functie de o coloana care sa fie afisata, dar in acest caz nu se mai poate folosi numarul de ordine al coloanei in loc de numele acesteia, inregistrarile vor fi sortate mai intai in functie de primul camp specificat dupa clauza ORDER BY, apoi, inregistarile care au aceeasi valoare in acest prim camp sunt sortate in functie de valoarea celui de-al doilea camp specificat dupa clauza ORDER BY, s.a.m.d.

De exemplu, pentru a sorta ascendent inregistrarile in functie de impozitul pe salariu folosim interogarea:

SQL> SELECT nume, salariu\*0.38

     FROM profesor

     ORDER BY salariu\*0.38;

care este echivalenta cu:

SQL> SELECT nume, salariu\*0.38

     FROM profesor

     ORDER BY 2;

NUME      SALARIU\*0.38

----      -----------

VOINEA    456

STANESCU 456

IONESCU   570

ALBU      836

MARIN          950

GEORGESCU     1064

GHEORGHIU     1140

Inregistrarile sunt sortate in mod implicit in ordine ascendenta (optiunea ASC), afisarea in ordine descendenta facandu-se prin utilizarea optiunii DESC. Observati ca in momentul sortarii valoarea Null este considerata cea mai mare, deci daca sortarea este ascendenta este trecuta pe ultima pozitie si daca sortarea este descendenta este trecuta pe prima pozitie. De exemplu:

SQL> SELECT grad, prima

FROM profesor

ORDER BY grad, prima DESC;

GRAD      PRIMA

----      -----

ASIST

ASIST      600

ASIST      150

CONF      200

LECT      2500

PROF

PROF      3500

Se observa ca in exemplul de mai sus inregistrarile au fost mai intai sortate ascendent (specificatie implicita) in functie de gradul didactic, inregistrarile cu acelasi grad au fost apoi ordonate in functie de cel de-al doilea criteriu de sortare, adica in functie de prima primita cu specificatia explicita de sortare descendenta.

**5.1.8. Clauza WHERE**

Clauza WHERE se foloseste pentru a regasi inregistrari ce corespund unei anumite conditii evaluata cu valoarea de adevar *True,*adica pentru a realiza anumite restrictii de selectie. Astfel, clauza WHERE corespunde restrictiilor operatorilor din algebra relationala. Cu alte cuvinte, daca o clauza ORDER BY este o clauza de sortare, clauza WHERE este o clauza de filtrare Daca nu se specifica o clauza WHERE, interogarea va intoarce ca rezultat toate randurile din tabel. Alaturi de clauza FROM care este obligatorie, WHERE este cea mai folosita clauza a comenzii SELECT. Din punct de vedere sintactic, clauza WHERE este optionala, dar atunci cand este introdusa urmeaza intotdeauna imediat dupa clauza FROM:

SELECT atribute

FROM obiect

WHERE conditie

Datorita existentei valorii Null, in SQL o conditie poate lua atat valorile *True si False*catsi valoarea*Necunoscut*(despre acest lucru se va discuta mai in detaliu in sectiunile urmatoare). O comanda SELECT cu clauza WHERE va returna toate inregistrarile pentru care conditia are valoarea *True.*Conditia clauzei WHERE poate cuprinde numele unor coloane, constante, operatori de comparatie sau operatori logici (NOT, AND, OR). Operatorii de comparatie se pot imparti in doua categorii: operatori relationali si operatori SQL. Toti acesti operatori sunt trecuti in revista in continuare.

**5.1.9. Operatori relationali**

Operatorii relationali sunt:

=          egal                              **<>**si != diferit

>          mai mare                      <            mai mic

>=        mai mare sau egal         <=         mai mic sau egal

Cele doua valori care sunt comparate trebuie sa apartina unor tipuri de date compatibile.

De exemplu, pentru a selecta toate cadrele didactice care nu apartin catedrei cu codul 10 folosim urmatoarea interogare:

SQL> SELECT nume, prenume

     FROM profesor

     WHERE cod\_catedra <>10;

NUME      PRENUME

----      -------

MARIN     VLAD

GEORGESCU CRISTIANA

ALBU      GHEORGHE

STANESCU  MARIA

Sirurile de caractere si data calendaristica trebuiesc incluse intre apostrofuri. De exemplu, pentru a selecta numai acele cadre didactice care au gradul didactic de profesor vom utiliza urmatoarea interogare:

SQL> SELECT nume, prenume

FROM profesor

WHERE grad ='PROF';

NUME      PRENUME

----      -------

GHEORGHIU STEFAN

MARIN     VLAD

In cazul sirurilor de caractere, literele mici sunt diferite de literele mari. De exemplu, urmatoarea interogare nu va returna nici o inregistrare:

SQL> SELECT nume, prenume

     FROM profesor

     WHERE grad='prof';

Toti operatorii de comparatie pot fi folositi atat pentru valori numerice cat si pentru siruri de caractere sau date calendaristice. De exemplu, pentru a afla toate cadrele didactice care s-au nascut inainte de l Ianuarie 1960 folosim interogarea:

SQL> SELECT nume,prenume, data\_nast

     FROM profesor

     WHERE data\_nast<'Ol-JAN-65';

NUME      PRENUME   DATA\_NAST

----      -------   ---------

GHEORGHIU STEFAN   ll-AUG-46

MARIN     VLAD      19-APR-45

GEORGESCU CRISTIANA 30-OCT-51

In cazul sirurilor de caractere ordonarea se face dupa codul ASCII al acestora. De exemplu, pentru a afla toate cadrele didactice ale caror nume sunt in ordinea alfabetica dupa litera 'M' se poate folosi interogarea.

SQL> SELECT nume, prenume

     FROM profesor

     WHERE nume>='M';

NUME      PRENUME

----      -------

MARIN     VLAD

VOINEA    MIRCEA

STANESCU MARIA

Trebuie remarcat ca interogarea de mai sus este corecta numai in cazul in care numele angajatilor incepe cu o litera mare, literele mici fiind in urma celor mari.

Exista posibilitatea de a compara valoarea unei coloane cu valoarea altei coloane pentru aceeasi inregistrare. De exemplu, daca dorim sa selectam acele cadre didactice care au primit prima mai mare decat salariul de baza vom avea:

SQL> SELECT nume, prenume, salariu, prima

     FROM profesor

     WHERE salariu<prima;

NUME      PRENUME   SALARIU   PRIMA

----      -------   -------   -----

GHEORGHIU     STEFAN   3000      3500

ALBU      GHEORGHE  2200      2500

**5.1.10. Operatori SQL**

Exista patru tipuri de operatori SQL, care pot opera cu toate tipurile de date:

1. BETWEEN. . .AND. . .

2. IN

3. LIKE

4. IS NULL

**Operatorul BETWEENAND**

Operatorul BETWEEN. . .AND. . . permite specificarea unui domeniu marginit de doua valori intre care trebuie sa se afle valoarea testata. Domeniul de valori specificat este un interval inchis iar limita inferioara trebuie specificata prima.

Astfel, daca dorim selectarea acelor cadre didactice care au salariul intre 2000 si 3000 vom folosi comanda:

SQL> SELECT nume, prenume, salariu

     FROM profesor

     WHERE salariu BETWEEN 2000 AND 3000;

NUME      PRENUME   SALARIU

----      -------   -------

GHEORGHIU STEFAN    3000

MARIN     VLAD       2500

GEORGESCU CRISTIANA  2800

ALBU      GHEORGHE  2200

**Operatorul IN**

Operatorul IN permite specificarea unei liste ele valori, valoarea testata trebuind sa se afle printre valorile acestei liste.

De exemplu, daca dorim selectarea cadrelor didactice care au gradul de conferentiar, lector sau asistent vom utiliza comanda:

SQL> SELECT nume, prenume, grad

     FROM profesor

     WHERE grad IN ('CONF', 'LECT', 'ASIST');

NUME      PRENUME   GRAD

----      -------   ----

GEORGESCU CRISTIANA CONF

IONESCU   VERONICA  ASIST

ALBU      GHEORGHE  LECT

VOINEA    MIRCEA    ASIST

STANESCU  MARIA     ASIST

**Operatorul LIKE**

Operatorul LIKE permite specificarea unui anumit model de sir de caractere cu care trebuie sa se potriveasca valoarea testata. Acest operator se foloseste in mod special atunci cand nu se stie exact valoarea care trebuie cautata. Pentru a construi modelul dupa care se face cautarea pot fi folosite doua simboluri:

% semnifica orice secventa de zero sau mai multe caractere

- semnifica orice caracter (care apare o singura data)

De exemplu, urmatoarea comanda SELECT va re turna toate cadrele didactice al caror nume incepe cu litera 'G':

SQL> SELECT nume, prenume

     FROM profesor

     WHERE nume LIKE 'G%';

NUME      PRENUME

----      -------

GHEORGHIU GEORGESCU

STEFAN    CRISTIANA

Daca dorim selectarea acelor cadre didactice al caror nume are litera 'O' pe a doua pozitie, indiferent de lungimea cuvantului, vom avea:

SQL> SELECT nume, prenume

     FROM profesor

     WHERE nume LIKE '\_0%';

NUME      PRENUME

----      -------

IONESCU   VERONICA

VOINEA    MIRCEA

O problema intervine atunci cand sirul contine caracterele % sau \_ (de exemplu sirul 'J\_James') deoarece aceste caractere au semnificatie predefmita. Pentru a schimba interpretarea acestor caractere se foloseste optiunea ESCAPE.

De exemplu, pentru a cauta toate titlurile de carte care incep cu caracterele 'J\_' se poate folosi interogarea:

SQL> SELECT titlu

     FROM carte

     WHERE titlu LIKE 'J/\_\_% ' ESCAPE '/';

In exemplul de mai sus optiunea ESCAPE identifica caracterul '/' ca fiind caracterul 'escape'. Deoarece in modelul folosit pentru LIKE acest caracter precede caracterul '\_' acesta din urma va fi interpretat ca o simpla litera, fara alta semnificatie.

Avantajul unei viteze mari de regasire ca urmare a indexarii este pierdut in momentul in care se cauta un sir de caractere care incepe cu \_ sau % intr-o coloana indexata.

Operatorul IS NULL

Operatorul IS NULL testeaza daca o valoare este Null. Pentru a vedea utilitatea acestui operator sa consideram urmatoarele interogari:

SQL> SELECT nume, prenume

     FROM profesor

     WHERE prima = NULL;

SQL> SELECT nume, prenume

     FROM profesor

     WHERE prima <> NULL;

Amandoua aceste interogari nu vor returna nici o inregistrare. Aceste lucru pare surprinzator la prima vedere deoarece ne-am fi asteptat ca prima interogare sa returneze toate cadrele didactice care nu au primit prima, iar a doua toate cadrele didactice care au primit prima. In SQL insa, orice conditie care este formata dintr-un operator de comparatie care are unul dintre termeni valoarea Null va avea ca rezultat valoarea Necunoscut, diferita de valoarea True (pentru care se face filtrarea). Pentru compararea cu Null se foloseste operatorul special IS NULL.

Deci pentru a afla cadrele didactice care nu au primit prima se foloseste interogarea:

SQL> SELECT nume, prenume FROM professor WHERE prima IS NULL;

NUME      PRENUME

----      -------

MARIN     VLAD

IONESCU  VERONICA

La fel, pentru a afla cadrele didactice ale caror data de nastere nu se cunoaste vom folosi urmatoarea interogare:

SQL> SELECT nume, prenume

     FROM profesor

     WHERE data\_nast IS NULL;

NUME      PRENUME

----      -------

IONESCU   ALBU

VERONICA GHEORGHE

**5.1.11. Operatorii logici**

Negarea operatorilor in unele cazuri suni mai usor de cautat inregistrarile care *nu*indeplinesc o anumita conditie. Acest lucru se poate realiza folosind operatorul NOT. Operatorul NOT se poate folosi pentru negarea unei expresii logice (de exemplu expresii de tipul NOT coloana = . . . ) sau pentru negarea operatorilor SQL in modul urmator:

        NOT BETWEEN . AND .

        NOT IN

        NOT LIKE

        IS NOT NULL

De exemplu, pentru a selecta cadrelor didactice al caror nume nu incepe cu litera 'G' se foloseste interogarea:

SQL> SELECT nume, prenume

     FROM profesor

     WHERE nume NOT LIKE 'G%';

NUME      PRENUME

----      -------

MARIN           VLAD

IONESCU       VERONICA

ALBU              GHEORGHE

VOINEA         MIRCEA

STANESCU    MARIA

Pentru a selecta cadrele didactice care au primit prima se foloseste interogarea:

SQL> SELECT nume, prenume

     FROM profesor

     WHERE prima IS NOT NULL;

NUME      PRENUME

----      -------

GHEORGHIU STEFAN

GEORGESCU CRISTIANA

ALBU      GHEORGHE

VOINEA    MIRCEA

STANESCU MARIA

Nota: Negarea unei expresii logice care are valoarea Necunoscut va avea tot valoare Necunoscut. De exemplu, o expresie de genul

NOT coloana = NULL

va avea valoarea Necunoscut, urmatoarea interogare nereturnand deci nici o inregistrare:

SQL> SELECT nume, prenume

     FROM profesor

     WHERE NOT prima = NULL;

**Conditii multiple de interogare (operatorii AND si OR)**

Operatorii AND si OR pot fi utilizati pentru a realiza interogari ce contin conditii multiple. Expresia ce contine operatorul AND este adevarata atunci cand ambele conditii sunt adevarate iar expresia ce contine operatorul OR este adevarata atunci cand cel putin una din conditii este adevarata. In aceeasi expresie logica se pot combina operatorii AND si OR dar operatorul AND are o precedenta mai mare decat operatorul OR, deci este evaluat mai intai.

In momentul evaluarii unei expresii, se calculeaza mai intai operatorii in ordinea precedentei, de la cel cu precedenta cea mai mare pana la cel cu precedenta cea mai mica. Daca operatorii au precedenta egala atunci ei sunt calculati de la stanga la dreapta.

Precedenta operatorilor, pornind de la cea mai mare la cea mai mica este urmatoarea:

        toti    operatorii    de    comparatie   si    operatorii    SQL    >,    <,    <=,    >=,    =,    <>, BETWEEN. . .AND. . ., IN, LIKE, IS   NULL;

        operatorul NOT;

        operatorul AND;

        operatorul OR.

Pentru a schimba prioritatea operatorilor se folosesc parantezele.

In exemplele de mai jos se observa modul de evaluare a expresiei in functie de precedenta operatorilor, precum si modul in care parantezele pot schimba acest lucru.

SQL> SELECT nume, prenume, salariu, cod\_catedra

     FROM profesor

     WHERE salariu>2000 AND cod\_catedra=10 OR cod\_catedra=20;

este echivalenta cu:

SQL> SELECT nume, prenume, salariu, cod\_catedra

     FROM profesor

     WHERE (salariu>2000 AND cod\_catedra=10) OR cod catedra=20;

NUME      PRENUME   SALARIU   COD-CATEDRA

----      -------   -------   -----------

GHEORGHIU STEFAN   3000      10

MARIN     VLAD      2500      20

ALBU      GHEORGHE  2200      20

STANESCU  MARIA     1200      20

dar este diferita de:

SQL> SELECT nume, prenume, salariu, cod\_catedra

     FROM profesor

     WHERE salariu>2000 AND (cod\_catedra=10 OR cod catedra=20);

NUME      PRENUME  SALARIU  COD-CATEDRA

----      -------   -------   -----------

GHEORGHIU STEFAN   3000      10

MARIN     VLAD      2500      20

ALBU      GHEORGHE  2200      20

**5.1.12. Functii**

Functiile sunt o caracteristica importanta a SQL si sunt utilizate pentru a realiza calcule asupra datelor, a modifica date, a manipula grupuri de inregistrari, a schimba formatul datelor sau pentru a converti diferite tipuri de date. Functiile se clasifica in doua tipuri:

1. Functii referitoare la o singura inregistrare:

        functii caracter;

        functii numerice;

        functii pentru data calendaristica si ora;

        functii de conversie;

        functii diverse.

2. Functii referitoare la mai multe inregistrari:

        functii totalizatoare sau functii de grup.

Diferenta dintre cele doua tipuri de functii este numarul de inregistrari pe care actioneaza: Functiile referitoare la o singura inregistrare returneaza un singur rezultat pentru fiecare rand al tabelului, pe cand functiile referitoare la mai multe inregistrari returneaza un singur rezultat pentru fiecare grup de inregistrari din tabel.

O observatie importanta este faptul ca daca se apeleaza o functie SQL ce are un argument egal cu valoarea Null, atunci in mod automat rezultatul va avea valoarea Null. Singurele functii care nu respecta aceasta regula sunt: CONCAT, DECODE, DUMP, NVL si REPLACE.

In continuare vom exemplifica si prezenta, la modul general, cele mai importante functii.

**5.1.13. Functii referitoare la o singura inregistrare**

Sunt functii utilizate pentru manipularea datelor individuale. Ele pot avea unul sau mai multe argumente si returneaza o valoare pentru fiecare rand rezultat in urma interogarii.

**Functii caracter**

Aceste functii au ca argumente date de tip caracter si returneaza date de tip VARCHAR2, CHAR sau NUMBER.

Cele mai importante functii caracter sunt:

        CONCAT- returneaza un sir de caractere format prin concatenarea a doua siruri;

        LOWER - modifica toate caracterele in litere mici;

        UPPER  - modifica toate caracterele in litere mari;

        LENGTH - returneaza numarul de caractere dintr-un anumit camp;

        REPLACE - cauta intr-un sir de caractere un subsir iar daca il gaseste il va inlocui cu un alt sir de caractere;

        SUBSTR - returneaza un subsir de caractere avand o anumita lungime incepand cu o anumita pozitie;

        TRANSLATE - cauta intr-un primsir de caractere fiecare dintre caracterele specificate intr-un al 2-lea sir, caracterele gasite fiind inlocuite de cele specificate intr-un al 3-lea sir.

Exemplu de utilizare a functiei LENGTH:

SQL> SELECT LENGTH (nume)

     FROM profesor;

LENGTH(NUME)

------------

9

5

9

7

4

6

8

Spre deosebire de alte functii, functiile caracter pot fi imbricate pana la orice adancime. Daca functiile sunt imbricate ele sunt evaluate din interior spre exterior. Pentru a determina, de exemplu, de cate ori apare caracterul 'A' in campul nume vom folosi interogarea:

SQL> SELECT nume, LENGTH (nume)-LENGTH (TRANSLATE    (nume,'DA','D'))

FROM profesor;

NUME      'A'

----      ---

GHEORGHIU 0

MARIN     1

GEORGESCU 0

IONESCU   0

ALBU      1

VOINEA    1

STANESCU  1

Nota: in exemplul de mai sus, functia TRANSLATE (nume, 'DA', 'D') va cauta in coloana "nume" primul caracter (caracterul 'D') din cel de-al doilea argument al functiei (sirul de caractere 'DA') si il va inlocui cu primul caracter (adica tot cu caracterul 'D') din cel de-al treilea argument al functiei (sirul de caractere 'D'), apoi va cauta cel de-al doilea caracter, adica caracterul 'A', si il va sterge din campul nume deoarece acesta nu are caracter corespondent in cel de-al treilea argument al functiei. Am folosit acest artificiu deoarece sirul de caractere vid este echivalent cu valoarea Null, deci functia TRANSLATE (nume, 'A', ' ') ar fi inlocuit toate valorile campului "nume" cu valoarea Null.

Functii numerice sau aritmetice

Aceste functii au ca argumente date numerice si returneaza tot valori numerice. Marea majoritate a acestor functii au o precizie de 38 de zecimale (COS, EXP, LN, LOG, SIN SQRT, TAN au insa o precizie de 36 de zecimale).

Dintre cele mai importante functii amintim:

        ROUND    - rotunjeste valorile la un anumit numar de pozitii zecimale;

        TRUNC     - trunchiaza valorile la un anumit numar de pozitii zecimale;

        CEIL          - returneaza cel mai mic intreg mai mare sau egal cu o anumita valoare;

        FLOOR     - returneaza cel mai mare intreg mai mic sau egal cu o anumita valoare;

        SIGN        - returneaza valoarea -l daca valoarea argumentului primit este mai mica decat 0, returneaza valoarea l daca valoarea  argumentului  primit  este mai  mare decat 0 si  0 daca valoarea argumentului primit este egala cu 0;

        SQRT        - returneaza radacina patrata a argumentului primit;

        ABS           - returneaza valoarea absoluta a argumentului primit;

        POWER    - returneaza valoarea unui numar ridicat la o anumita putere;

        MOD         - returneaza restul impartirii a doua numere;

- alte functii matematice cum ar fi: LOG, SIN, TAN, COS, EXP, LN.

Functii pentru data calendaristica si ora

In Oracle datele de tip data calendaristica sunt reprezentate sub un format numeric reprezentand: ziua, luna, anul, ora, minutul, secunda si secolul. Oracle poate manevra date calendaristice de la l ianuarie 4712 i. Cr pana la 31 decembrie 4712 d. Cr. Modul implicit de afisare si introducere este sub forma: DD-MON-W (ex. '31-Dec-99'). Aceasta categorie de functii opereaza pe valori de tip data calendaristica, rezultatul returat fiind tot de tip data calendaristica, exceptie facand functia MONTHS BETWEEN care retureaza o valoare numerica.

Cele mai des intalnite functii sunt:

        ADD\_MONTH  - returneaza o data calendaristica formata prin adaugarea la data calendaristica specificata a unui anumit numar de luni;

        LAST\_DAY - intoarce ca rezultat ultima zi a unei luni specificate;

        MONTHS\_ BETWEEN - returneaza numarul de luni dintre doua date calendaristice specificate;

        NEXT\_DAY  - returneaza prima data calendaristica ulterioara datei calendaristice specificate;

        SYSDATE - intoarce ca rezultat data calendaristica a sistemului.

Asupra datelor calendaristice se pot realiza operatii aritmetice, cum ar fi scaderea sau adunarea, modul lor de functionare fiind ilustrat in tabelul de mai jos:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tip  operand | Operatie | Tip operand | Tip rezultat | Descriere |
| data | +/- | numar | data | Adauga/scade un numar de zile la o data calendaristica |
| data | +/- | numar/24 | data | Adauga/scade un numar de ore la o data calendaristica |
| data | +/- | numar/ 1440 | data | Adauga/scade un numar de minute la o data calendaristica |
| data | +/- | numar/86400 | data | Adauga/scade un numar de secunde la o data calendaristica |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tip  operand | Operatie | Tip operand | Tip rezultat | Descriere |
| data | \_ | data | numar  zile | Scade doua date calendaristice rezultand diferenta in numar de zile.Daca al doilea operand este mai mare decat primul numarul de zile rezultat este reprezentat de o valoare negativa |

De asemenea, mai exista functiile ROUND si TRUNC care rotunjesc, respectiv trunchiaza data calendaristica. Aceste functii sunt foarte folositoare atunci cand se doreste compararea datelor calendaristice care au ora diferita. Exemplul urmator rotunjeste data de nastere a cadrelor didactice in functie de an:

SQL> SELECT ROUND(data nast,'YEAR') 'DATA'

     FROM  profesor;

DATA

----

0l-JAN-47

0l-JAN-45

01-JAN 52

0l-JAN-66

0l-JAN-70

Functii de conversie

In general expresiile nu pot contine valori apartinand unor tipuri de date diferite. De exemplu, nu se poate inmulti 3 cu 7 si apoi aduna 'ION'. Prin urmare se realizeaza anumite conversii, care pot fi implicite sau explicite.

Conversiile implicite se realizeaza in urmatoarele cazuri:

        atribuiri de valori unei coloane (folosind comenzile INSERT sau UPDATE) sau atribuirilor de valori unor argumente ale unei functii;

        evaluari de expresii.

Pentru atribuiri, programul Oracle efectueaza in mod implicit urmatoarele conversii de tip:

        VARCHAR2 sau CHAR la NUMBER

        VARCHAR2 sau CHAR la DATE

        VARCHAR2 sau CHAR la ROWI D

        NUMBER,   DATE sau ROWI D la VARCHAR2

Conversia la atribuire reuseste in cazul in care Oracle poate converti tipul valorii atribuite la tipul destinatiei atribuirii.

Pentru evaluarea expresiilor, se realizeaza in mod implicit urmatoarele conversii de tip:

        VARCHAR2 sau   CHAR la NUMBER

        VARCHAR2 sau   CHAR la DATE

        VARCHAR2   sau CHAR la ROWI D

De exemplu, pentru urmatoarea interogare se realizeaza conversia in mod implicit a constantei de tip CHAR, '10', la tipul NUMBER.

SQL> SELECT salariu + '10'

     FROM profesor;

SALARIU+'10'

------------

3010

2510

2810

1510

2210

1210

1210

Pentru  conversiile explicite de  tip, SQL pune la dispozitie mai multe functii de conversie, de la un anumit tip de data la altul, dupa cum este aratat in tabelul de mai jos.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tip convertit  Tip initial | CHAR | NUMBER | DATE | RAW | ROWID |
| CHAR |  | TO\_NUMBER | TO\_DATE | HEXTORAW | CHARTOROWID |
| NUMBER | TO\_CHAR |  | TO\_DATE(nr,'J') |  |  |
| DATE | TO\_CHAR | TO\_DATE(data,'J') |  |  |  |
| RAW | RAWTOHEX | - |  |  |  |
| RAWID | RAWIDTOCHAR |  |  |  |  |

Cele mai uzuale functii sunt:

        TO\_ CHAR - converteste un numar sau o data calendaristica intr-un sir de caractere;

        T0\_ NUMBER - converteste un sir de caractere alcatuit din cifre intr-o valoare numerica;

        T0\_ DATE - converteste  un  sir  de  caractere sau  un  numar  ce  reprezinta  o  data calendaristica la o valoare de tip data calendaristica. De asemenea poate converti o data calendaristica la un numar ce reprezinta data calendaristica Iuliana.

Pentru a realiza conversia, aceste functii folosesc anumite masti de format.

Urmatorul exemplu va prelua data si ora curenta a sistemului din functia SYSDATE si o va formata intr-o data scrisa pe litere ce va contine si ora in minute si secunde:

SQL> SELECT TO\_CHAR(SYSDATE,'DD MONTH YYYY HH24:MI:SS') data   FROM dual;

DATA

----------------------------

17 MAY        2000 17:03:38

Functii diverse

Acestea sunt in general functii care accepta ca argumente orice tip de data. Cele mai utilizate sunt:

        DECODE - Aceasta este una dintre cele mai puternice functii SQL. Practic, aceasta faciliteaza interogarile conditionate, actionand ca o comanda 'if-then-else1 sau 'case' dintr-un limbaj procedural. Pentru fiecare inregistrare se va evalua valoarea din coloana testata si se va compara pe rand cu fiecare valoare declarata in cadrul functiei. Daca se gasesc valori egale, atunci functia va returna o valoare aferenta acestei egalitati, declarata tot in cadrul functiei. Se poate specifica ca, in cazul in care nu se gasesc valori egale, functia sa intoarca o anumita valoare. Daca acest lucru nu se specifica functia va intoarce valoarea Null.

        GREATEST - returneaza cea mai mare valoare dintr-o lista de valori;

        LEAST -returneaza cea mai mica valoare dintr-o lista de valori;

        VSIZE - returneaza numarul de bytes pe care este reprezentata intern o anumita coloana;

        USER - returneaza numele utilizatorului curent al bazei de date;

        DUMP - returneaza o valoare ce contine codul tipului de data, lungimea in bytes, precum si reprezentarea interna a unei expresii.

Exemplul urmator utilizeaza functia DECODE pentru a returna o crestere a salariului cadrelor didactice cu grad de profesor, conferentiar si lector, restul salariilor ramanand nemodificate:

SQL> SELECT nume, grad, salariu,     DECODE(grad,'PROF',salariu\*1.2,CONF,salariu\*1.15,

     'LECT', salariu\*1.1, salariu) 'Salariu modificat'

     FROM profesor;

NUME      GRAD SALARIU   Salariu  modificat

----      ---- -------   ------------------

GHEORGHIU PROF 3000         3600

MARIN     PROF 2500         3000

GEORGESCU CONF 2800         3220

IONESCU   ASIST 3500         1500

ALBU      LECT 2200         2420

VOINEA    ASIST 1200         1200

KKKKK

http://ib.adnxs.com/getuid?http%3a%2f%2fdis.criteo.com%2frex%2fmatch.aspx%3fc%3d11%26uid%3d%24UID

STANESCU  ASIST 1200         1200

**5.1.14. Functii referitoare la mai multe inregistrari**

Aceste funtii se mai numesc si functii totalizatoare sau functii de grup. Spre deosebire de functiile referitoare la o singura inregistrare, functiile de grup opereaza pe un set ele mai multe inregistrari si returneaza un singur rezultat pentru fiecare grup. Daca nu este utilizata clauza GROUP BY, ce grupeaza inregistrarile dupa un anumit criteriu, tabela este considerata ca un singur grup si se va returna un singur rezultat.

        COUNT - determina numarul de inregistrari care indeplinesc o anumita conditie;

        MAX - determina cea mai mare valoare dintr-o coloana;

        MIN - determina cea mai mica valoare dintr-o coloana;

        SUM - returneaza suma tuturor valorilor dintr-o coloana;

        AVG - calculeaza valoarea medie a unei coloane;

        STDDEV - determina abaterea sau deviatia standard a unei coloane numerice;

        VARIANCE  - returneaza dispersia, adica patratul unei deviatii standard pentru o  coloana numerica.

De exemplu:

SQL> SELECT MIN(salariu.),MAX(salariu),AVG(salariu),COUNT(\*) FROM profesor;

MIN(SALARIU)  MAX(SALARIU)  AVG(SALARIU)  COUNT(\*)

------------ ------------  ------------  --------

1200         3000           2057.1429     7

Toate functiile de mai sus, cu exceptia functiei COUNT, opereaza asupra unei coloane sau unei expresii, care este specificata ca parametri al functiei. In cazul functiei COUNT, argumentul acesteia nu conteaza, de obicei utilizandu-se ca argument simbolul \*.

Nota: Toate functiile de mai sus ignora valorile Null, exceptie facand functia COUNT. Pentru a include in calcule si inregistrarile cu valoarea Null se poate folosi functia NVL.

Daca nu este utilizata clauza GROUP BY, in lista de atribute ale comenzii SELECT nu pot apare functii de grup alaturi de nume de coloane sau alte expresii care iau valori pentru fiecare inregistrare in parte. De exemplu, urmatoarea interogare va genera o eroare:

SQL> SELECT nume, KIN(salariu)

FROM profesor;

ERROR at line 1:

ORA-00937: not a single-group group function

**5.1.15 Pseudo-coloana ROWNUM**

ROWNUM este o pseudo-coloana care numeroteaza randurile selectate de o interogare. Astfel, pentru primul rand selectat pseudo-coloana ROWNUM are valoarea l, pentru al doilea rand are valoarea 2, s.a.m.d. De exemplu, pentru a limita randurile selectate de o interogare la maxim 5, se foloseste urmatoarea comanda:

     SELECT\*

     FROM profesor

     WHERE ROWNUM<6;

Deoarece pscudo-coloana ROWNUM numeroteaza randurile selectate, valorile sale vor trebui sa inceapa tot timpul cu 1. Deci daca in exemplul de mai sus conditia ar fi ROWNUM > 6 sau ROWNUM = 6 interogarea nu ar selecta nici un rand deoarece in acest caz conditia ar fi intotdeauna falsa. Pentru primul rand accesat de interogare ROWNOM va avea valoarea l, conditia nu este indeplinita si deci randul nu va fi selectat. Pentru al doilea rand accesat de interogare ROWNUM va avea din nou valoarea l, conditia nu este indeplinita nici in acest caz si deci nici acest rand nu va fi selectat, s.a.m.d. Prin urmare nici unul din randurile accesate nu vor satisface conditia continuta de interogare.

Pseudo-coloana ROWNUM se poate folosi atat in conditia unor comenzi UPDATE sau DELETE cat si pentru a atribui valori unice unei coloane, ca in exemplul de mai jos:

        UPDATE vanzari

 SET cod = ROWNUM;

Valoarea pseudo-coloanei ROWNUM este atribuita unui rand inainte ca acesta sa fie sortat datorita unei clauze ORDER BY, de aceea valorile pseudo-coloanei nu reprezinta ordinea de sortare. De exemplu:

SQL> SELECT  nume,   prenume,   ROWNUM

     FROM  profesor

     ORDER  BY   salariu;

NUME      PRENUME   ROWNUM

----      -------   ------

VOINEA    MIRCEA        6

STANESCU  MARIA         7

IONESCU   VERONICA       4

ALBU      GHEORGHE       5

MARIN     VLAD           2

GEORGESCU CRISTIANA      3

GHEORGHIU STEFAN        1

**5.1.16. Clauza GROUP BY**

Clauza GROUP BY este utilizata pentru a imparti din punct de vedere logic un tabel in grupuri de inregistrari. Fiecare grup este format din toate inregistrarile care au aceeasi valoare in campul sau grupul de campuri specificate in clauza GROUP BY. Unele inregistrari pot fi excluse folosind clauza WHERE inainte ca tabelul sa fie impartit in grupuri.

Clauza GROUP BY se foloseste de obicei impreuna cu functiile de grup, acestea returnand valoarea calculata pentru fiecare grup in parte. In cazul folosirii clauzei GROUP BY, toate expresiile care apar in lista atributelor comenzii SELECT trebuie sa aiba o valoare unica pentru fiecare grup, de aceea orice coloana sau expresie din aceasta lista care nu este o functie de grup trebuie sa apara in clauza GROUP BY.

SQL> SELECT grad, AVG (salariu)

     FROM profesor

     GROUP BY grad;

GRAD         AVG(SALARIU)

----         -----------

ASIST         1300

CONF          2800

LECT          2200

PROF          2750

SQL> SELECT grad, MAX(salariu)

     FROM profesor

     WHERE prima IS NOT NULL

     GROUP BY grad;

GRAD      MAX(SALARIU)

----      ------------

ASIST         1200

CONF          2800

LECT          2200

PROF          3000

Urmatoarea interogare va genera o eroare deoarece in lista atributelor comenzii SELECT exista o coloana (nume) care nu apare in clauza GROUP BY:

SQL >SFLECT nume, MIN (salariu)

     FROM profesor

     GROUP BY grad;

ERROR at line 1:

ORA-00979: not a GROUP BY expression

Comanda de mai sus este invalida deoarece coloana "nume" are valori individuale pentru fiecare inregistrare, in timp ce MIN (salariu) are o singuri valoare pentru un grup.

Clauza GROUP BY permite apelarea unei functii de grup in alta functie de grup. In exemplul urmator, functia AVG retumeaza salariul mediu pentru fiecare grad didactic, iar functia MAX remrneaza maximul dintre aceste salarii medii.

SQL> SELECT MAX(AVG(salariu))

     FROM profesor

     GROUP BY grad;

**5.1.17. Clauza HAVING**

Clauza HAVING este tot o clauza de filtrare ca si clauza WHERE. Totusi, in timp ce clauza WHERE determina ce inregistrari vor fi selectionate dintr-un tabel, clauza HAVING determina care dintre grupurile rezultate vor fi afisate dupa ce inregistrarile din tabel au fost grupate cu clauza GROUP BY. Cu alte cuvinte, pentru a exclude grupuri de inregistrari se foloseste clauza HAVING iar pentru a exclude inregistrari individuale se foloseste clauza WHERE.

Clauza HAVING este folosita numai daca este folosita si clauza GROUP BY. Expresiile folosite intr-o clauza HAVING trebuie sa aiba o singura valoare pe grup.

Atunci cand se foloseste clauza GROUP BY, clauza WHERE se utilizeaza pentru eliminarea inregistrarilor ce nu se doresc a fi grupate. Astfel, urmatoarea interogare este invalida deoarece clauza WHERE incearca sa excluda giupuri de inregistrari si nu anumite inregistrari:

SQL> SELECT grad, AVG(salariu)

     FROM profesor

WHERE AVG(salariu)>2000

GROUP BY grad;

ERROR:

ORA-00934: group function is not allowed here

Pentru a exclude gradul didactic pentru care media de salariu nu este mai mare decat 2000 se foloseste urmatoarea comanda SELECT cu clauza HAVING:

SQL> SELECT grad, AVG(salariu)

     FROM profesor

     GROUP BY grad

     HAVING AVG(salariu)>2000;

GRAD AVG(SALARIU)

---- ------------

CONF 2800

LECT 2200

PROF 2750

Urmatoarea interogare exclude intai cadrele didactice care nu au salariu mai mare decat 2500 dupa care exclude gradul didactic pentru care media de salariu nu este mai mare decat 2800.

SQL> SELECT grad, AVG(salariu)

     FROM profesor

     WHERE salariu > 2500

     GROUP BY grad

     HAVING AVG(salariu) > 2800;

GRAD      AVG(SALARIU)

----      -----------

PROF      3000

**5.1.18. Regasirea datelor din doua sau mai multe tabele**

O jonctiune este o interogare care regaseste inregistrari din doua sau mai multe tabele. Capacitatea de a realiza o jonctiune intre doua sau mai multe tabele reprezinta una dintre cele mai puternice facilitati ale unui sistem relational. Legatura dintre inregistrarile tabelelor se realizeaza prin existenta unor campuri comune caracterizate prin domenii de definitie compatibile (chei primare sau straine). Pentru realizarea unei jonctiuni se foloseste comanda SELECT, precizand in clauza FROM numele tabelelor utilizate, iar in clauza WHERE criteriul de compunere.

**Produsul a doua sau mai multe tabele.**

In cazul in care in interogare se specifica mai multe tabele si nu este inclusa o clauza WHERE, interogarea va genera produsul cartezian al tabelelor. Acesta va contine toate combinatiile posibile de inregistrari din tabelele componente. Astfel, produsul cartezian a doua tabele care contin 100, respectiv 50 de inregistrari va avea dimensiunea de 5.000 de inregistrari.

De exemplu, sa consideram tabela **CATEDRA** cu urmatoarele 4 inregistrari:

COD\_CATEDRA   NUME           PROFIL

-----------   ----------    ------

10             INFORMATICA   TEHNIC

20             ELECTRONICA   TEHNIC

30             AUTOMATICA    TEHNIC

40             FINANTE       ECONOMIC

Atunci urmatoarea interogare va genera produsul cartezian al tabelelor, adica va avea ca rezultat 7 x 4*=*28 de randuri ce vor contine toate combinatiile posibile de inregistrari din cele doua tabele:

SOL> SELECT \*

     FROM profesor, catedra;

Daca in lista de atribute ale comenzii SELECT sunt specificate coloanele selectate, atunci numele acestora trebuie sa fie unice in cadrul tuturor tabelelor. Daca exista un nume de coloana care apare in mai mult de un tabel, atunci, pentru evitarea ambiguitatii, trebuie specificat si tabelul din care face parte coloana in cauza. De exemplu, in urmatoarea interogare pentru coloanele "cod\_catedra" si "nume" trebuie specificate tabelele din care fac parte:

SQL> SELECT profesor.nume, prenume, catedra.cod\_catedra, catedra.nume

     FROM profesor, catedra;

NUME      PRENUME   COD CATEDRA   NUME

----      -------   -----------   ----

GHEORGHIU STEFAN    10             INFORMATICA

MARIN     VLAD      10             INFORMATICA

GEORGESCU CRISTIANA 10             INFORMATICA

IONESCU   VERONICA  10             INFORMATICA

ALBU      GHEORGHE  10             INFORMATICA

VOINEA    MIRCEA    10             INFORMATICA

STANESCU  MARIA     10             INFORMATICA

GHEORGHIU STEFAN    20             ELECTRONICA

MARIN     VLAD      20             ELECTRONICA

GEORGESCU CRISTIANA 20             ELECTRONICA

IONESCU   VERONICA  20             ELECTRONICA

ALBU      GHEORGHE  20             ELECTRONICA

VOINEA    MIRCEA    20             ELECTRONICA

STANESCU  MARIA     20             ELECTRONICA

GHEORGHIU STEFAN    30             AUTOMATICA

MARIN     VLAD      30             AUTOMATICA

GEORGESCU CRISTIANA 30             AUTOMATICA

IONESCU   VERONICA  30             AUTOMATICA

ALBU      GHEORGHE  30             AUTOMATICA

VOINEA    MIRCEA    30             AUTOMATICA

STANESCU  MARIA     30             AUTOMATICA

GHEORGHIU STEFAN    40             FINANTE

MARIN     VLAD      40             FINANTE

GEORGESCU CRISTIANA 40             FINANTE

IONESCU   VERONICA  40             FINANTE

ALBU      GHEORGHE  40             FINANTE

VOINEA    MIRCEA    40             FINANTE

STANESCU  MARIA     40             FINANTE

In general, pentru a scurta textul comenzii, in astfel de cazuri se folosesc de obicei alias-uri pentru numele tabelelor, care pot fi folosite in interogare. Astfel interogarea de mai sus se mai poate scrie:

SQL> SELECT p.nume, prenume, c.cod\_catedra, c.nume

     FROM profesor p,catedra c;

In general, produsul cartezian este rar folosit, avand o utilitate practica redusa.

Jonctiuni

Pentru a realiza o jonctiune intre doua sau mai multe tabele se utilizeaza clauza WHERE a interogarilor pe aceste tabele, in functie de criteriul de compunere, se disting mai multe tipuri de jonctiuni:

1.      jonctiuni echivalente (EQUI-JOIN) sau jonctiuni interne (INNER JOIN)

2.      jonctiuni neechivalente

3.      jonctiuni externe (OUTER JOIN)

4.      autojonctiuni

**1. Jonctiunile echivalente**

O echijonctiune contine operatorul egalitate (=) in clauza WHERE, combinand inregistrarile din tabele care au valori egale pentru coloanele specificate.

De exemplu, pentru a afisa cadrele didactice si numele catedrei din care acestea fac parte se combina inregistrarile din cele doua tabele pentru care codul catedrei este acelasi.

SQL> SELECT p.nume, p.prenume, c.nume 'NUME CATEDRA'

     FROM profesor p, catedra c

     WHERE p.cod catedra=c.cod catedra;

NUME      PRENUME   NUME CATEDRA

----      -------   ------------

GHEORGHIU STEFAN   INFORMATICA

IONESCU   VERONICA  INFORMATICA

VOINEA    MIRCEA    INFORMATICA

MARIN     STANESCU  ELECTRONICA

ALBU      GEORGESCU ELECTRONICA

VLAD      MARIA     ELECTRONICA

GHEORGHE CRISTIANA AUTOMATICA

**2. Jonctiuni neechivalente**

Jonctiunile neechivalente sunt acelea care nu folosesc in clauza WHERE operatorul egal. Operatorii cei mai utilizati in cazul jonctiunilor neechivalente sunt: <, >, <=,   >=, <>, BETWEENAND.

Pentru a exemplifica un astfel de tip de jonctiune consideram tabela gradsal ce contine pragul minim si pragul maxim al salariului dintr-un anumit grad de salarizare:

GRAD SALARIZARE    PRAG MIN  PRAG MAX

---------------    --------  --------

1                  500       1500

2                  1501      2000

3                  2001      2500

4                  2501      3500

5                  3501      10000

Evident, intre tabelele *profesor* si *gradsal* nu are sens definirea unei jonctiuni echivalente deoarece nu exista o coloana din tabela *profesor* careia sa-i corespunda o coloana din tabela *gradsal*. Exemplul urmator ilustreaza definirea unei jonctiuni neechivalente care evalueaza gradul de salarizare a cadrelor didactice, prin incadrarea salariului acestora intr-un interval stabilit de pragul minim si pragul maxim:

SQL> SELECT p.nume, p.grad, p.salariu, g.grad\_salarizare

     FROM profesor p, gradsal g

     WHERE p.salariu BETWEEN g.prag\_min AND g.prag\_max;

NUME      GRAD      SALARIU   GRAD SALARIZARE

----      ----      -------   ---------------

IONESCU   ASIST     1500           1

VOINEA    ASIST     1200           1

STANESCU  ASIST     1200           1

MARIN     PROF      2500           3

ALBU      LECT      2200           3

GHEORGHIU PROF      3000           4

GEORGESCU CONF      2800           4

**3. Jonctiuni externe**

Daca intr-o jonctiune de tipul celor prezentate pana acum una sau mai multe inregistrari nu satisfac conditia de compunere specificata in clauza WHERE, atunci ele nu vor aparea in rezultatul interogarii. Aceste inregistrari pot apare insa daca se foloseste jonctiunea externa. Jonctiunea externa returneaza toate inregistrarile care satisfac conditia de jonctiune plus acele inregistrari dintr-un tabel ale carorvalori din coloanele dupa care se face legatura nu se regasesc in coloanele corespunzatoare ale nici unei inregistrari din celalalt tabel.

Pentru a realiza o jonctiune externa intre tabelele A si B ce returneaza toate inregistrarile din tabela A se utilizeaza seninul ( + ) in dreapta tabelului B. Pentru fiecare inregistrare din tabela A care nu satisface conditia de compunere pentru nici o inregistrare din tabela B, se va crea in tabela B o inregistrare nula care va fi compusa cu inregistrarea din tabela A. Invers, pentru a realiza o jonctiune externa intre tabelele A si B ce returneaza toate inregistrarile din tabela B, se utilizeaza semnul ( + ) in dreapta tabelului A.

In interogarea utilizata pentru a exemplifica jonctiunea echivalenta, se observa ca au fost selectate numai catedrele in care exista cadre didactice. Pentru a afisa toate catedrele, indiferent daca ele cuprind sau nu cadre didactice, se foloseste urmatoarea interogare:

SQL> SELECT p.nume, p.prenume, c.nume

     FROM profesor p, catedra c

     WHERE p.cod catedra(+)=c.cod catedra;

NUME      PRENUME   NUME

----      -------   ----

GHEORGIU  STEFAN    INFORMATICA

IONESCU   VERONICA  INFORMATICA

VOINEA     MIRCEA     INFORMATICA

MARIN     VLAD      ELECTRONICA

STANESCU  MARIA         ELECTRONICA

ALBU       GHEORGHE  ELECTRONICA

GEORGESCU CRISTIANA AUTOMATICA

FINANTE

Se observa ca ultima inregistrare (ce corespunde catedrei de finante care nu are in componenta niciun cadru didactic) va avea coloanele corespunzatoare primului tabel completate cu Null.

Folosirea operatorului de jonctiune externa are urmatoarele restrictii:

1.      Operatorul ( + ) poate fi plasat in oricare parte a conditiei din clauza WHERE, insa nu in ambele parti. Tabelul de partea caruia este amplasat acest operator va crea inregistrari nule care vor fi compuse cu inregistrarile din celalalt tabel care nu satisfac conditia de compunere.

2.      Daca tabelele A si B au conditii multiple de jonctiune, atunci operatorul (+) trebuie utilizat in toate aceste conditii.

3.      Intr-o singura interogare nu se poate realiza o jonctiune externa a unui tabel cu mai multe tabele.

4.      O conditie care contine operatorul (+) nu poate fi combinata cu o alta conditie ce utilizeazaoperatorul IN.

5.      O conditie care contine operatorul ( + ) nu poate fi combinata cu o alta conditie prin operatorul OR.

**5.**      **Auto-jonctiuni**

Auto-jonctiunea reprezinta jonctiunea unui tabel cu el insusi. Pentru ca randurile dintr-un tabel sa poata fi compuse cu randuri din acelasi tabel, in clauza FROM a interogarii numele tabelului va apare de mai multe ori, urmat de fiecare data de un alias.

De exemplu, pentru a selecta toate cadrele didactice care au un sef direct si numele acestui sef se foloseste urmatoarea auto-jonctiune:

SQL> SELECT p.nume, p.prenume, s.nume, s.prenume

     FROM profesor p, profesor s

     WHERE p.sef=s.cod;

NUME           PRENUME        NUME           PRENUME

----           -------        ---------     -------

MARIN         VLAD           GHEORGHIU      STEFAN

GEORGESCU     CRISTIANA      GHEORGHIU      STEFAN

ALBU          GHEORGHE       GHEORGHIU      STEFAN

VOINEA          MIRCEA          GHEORGHIU      STEFAN

IONESCU       VERONICA       GEORGESCU      CRISTIANA

STANESCU      MARIA         IONESCU        VERONICA

Autojonctiunea poate fi folosita si pentru verificarea corectitudinii interne a datelor. De exemplu, esteputin probabil sa existe doua cadre didactice care au cod diferit dar in schimb au acelasi nume, prenume si data de nastere. Pentru a verifica daca exista astfel de inregistrari se foloseste interogarea:

SQL> SELECT a.nume, a.prenume

     FROM profesor a, profesor b

     WHERE a.nume=b.nume AND a.prenume=b.prenume AND a.data-  nast= b.data-nast AND a.cod<>b.cod

**5.1.19.**Operatorii pentru multimi

Operatorii de multimi combina doua sau mai multe interogari, efectuand operatii specificemultimilor: reuniune, intersectie, diferenta. Acesti operatori se mai numesc si *operatori verticali*deoarece combinarea celor doua interogari se face coloana cu coloana. Din acest motiv, numarul total de coloane si tipurile de date ale coloanelor coresondente din cele doua interogari trebuie sa coincida:

Exista urmatorii operatori pentru multimi:

1.      UNION - Returneaza rezultatele a doua sau mai multe interogari eliminind toate inregistrarile duplicat;

2.      UNION ALL - Returneaza rezultatele a doua sau mai multe interogari incluzand inregistrarile duplicat;

3.      INTERSECT - Returneaza toate inregistrarile distincte gasite in ambele interogari;

4.      MINUS - Returneaza toate inregistrarile distincte care se gasesc in prima interogare dar nu in a doua interogare.

Sa consideram de exemplu urmatoarele interogari:

SQL> SELECT grad, salariu

     FROM profesor

     WHERE cod catedra = 10;

GRAD      SALARIU

----      -------

PROF      3000

ASIST     1500

ASIST     1200

SQL> SELECT grad, salariu

     FROM profesor

     WHERE cod catedra = 20;

GRAD      SALARIU

----      -------

PROF      2500

LECT      2200

ASIST     1200

In continuare exemplificam fiecare dintre operatorii pentru multimi aplicati acestor interogari:

SQL> SELECT grad, salariu

     FROM profesor

     WHERE cod\_catedra = 10

     UNION

     SELECT grad, salariu

     FROM profesor

     WHERE cod\_catedra = 20

ORDER BY salariu;

GRAD      SALARIU

----      -------

ASIST     1200

ASIST     1500

LECT      2200

PROF      2500

PROF      3000

SQL> SELECT grad, salariu

     FROM profesor

     WHERE cod\_ catedra = 10

     UNION ALL

     SELECT grad, salariu

     FROM profesor

     WHERE cod\_catedra = 20;

GRAD      SALARIU

----      -------

PROF      3000

ASIST     1500

ASIST     1200

PROF      2500

LECT      2200

ASIST     1200

SQL> SELECT grad, salariu

     FROM profesor

     WHERE cod\_catedra = 10

     INTERSECT

     SELECT grad, salariu

     FROM profesor

     WHERE cod\_catedra = 20;

GRAD      SALARIU

----      -------

ASIST     1200

SQL> SELECT grad, salariu

     FROM profesor

     WHERE cod catedra = 10

     MINUS

     SELECT grad, salariu

     FROM profesor

     WHERE cod catedra = 20;

GRAD      SALARIU

----      -------

ASIST     1500

PROF      3000

Exista urmatoarele reguli de folosire a operatorilor pentru multimi:

        interogarile trebuie sa contina acelasi numar de coloane;

        coloanele corespondente trebuie sa aiba acelasi tip de data;

        in rezultat vor aparea numele coloanelor din prima interogare, nu cele din a doua interogare chiar daca aceasta foloseste alias-uri, de exemplu:

SQL> SELECT cod

     FROM profesor

     MINUS

     SELECT sef

     FROM profesor;

COD

---

101

104

105

106

        clauza ORDER BY poate fi folosita o singura data intr-o interogare care foloseste operatori de multimi; atunci cand se foloseste, ea trebuie pozitionala la sfarsitul comenzii; de exemplu:

SQL> SELECT grad, salariu

     FROM profesor

     WHERE cod\_catedra = 10

     UNION

     SELECT grad, salariu

     FROM profesor

     WHERE cod catedra = 20

     ORDER BY 2;

GRAD      SALARIU

----      -------

ASIST     1200

ASIST     1500

LECT      2200

PROF      2500

PROF      3000

.    operatorii pentru multimi pot fi utilizati in subinterogari;

.    pentru a modifica ordinea de executie este posibila utilizarea parantezelor, de exemplu:

SQL> SELECT grad

     FROM profesor

     WHERE cod-catedra = 10

     INTERSECT

     SELECT grad

     FROM profesor

     WHERE cod\_catedra = 20

     UNION

     SELECT grad

     FROM profesor

     WHERE cod\_catedra = 30;

GRAD

----

ASIST

CONF

PROF

SQL> SELECT grad

     FROM profesor

     WHERE cod\_catedra = 10

     INTERSECT

     (SELECT grad

     FROM profesor

     WHERE cod\_catedra = 20

     UNION

     SELECT grad

     FROM profesor

     WHERE cod\_catedra = 30);

GRAD

----

ASIST

PROF

**5.1.20. Subinterogari si operatorii ANY, ALL, EXISTS**

O *subinterogare*este o comanda SELECT inclusa in alta comanda SELECT. Rezultatele subinterogarii sunt transmise celeilalte interogari si pot aparea in cadrul clauzelor WHERE, HAVING sau FROM.Subinterogarile sunt utile pentru a scrie interogari bazate pe o conditie in care valoarea de comparatie este necunoscuta. Aceasta valoare poate fi aflata folosind o subinterogare. De exemplu:

SELECT coloane

FROM tabel

WHERE coloana = (SELECT coloane

    FROM tabel

    WHERE conditie).

Subinterogarea, denumita si *interogare interioara (inner query),*genereaza valorile pentru conditia de cautare a instructiunii SELECT care o contine, denumita *interogare exterioara (outer query).*Instructiunea SELECT exterioara depinde de valorile generate de catre interogarea interioara. In general, interogarea interioara se executa prima si rezultatul acesteia este utilizat in interogarea exterioara. Rezultatul interogarii exterioare depinde de numarul valorilor returnate de catre interogarea interioara. In acest sens, putem distinge:

*1.*      *Subinterogari care returneaza  un singur rand;*

2.      *Subinterogari care returneaza mai multe randuri.*

Din punct de vedere al ordinii de evaluare a interogarilor putem clasifica subinterogarile in:

1.      *Subinterogari simple -*in care interogarea interioara este evaluata prima, independent de interogarea exterioara (interogarea interioara se executa o singura data);

2.      *Subinterogari corelate -*in care valorile returnate de interogarea interioara depind de valorile returnate de interogarea exterioara (interogarea interioara este evaluata pentru fiecare inregistrare a interogarii exterioare).

Subinterogarile sunt indeosebi utilizate atunci cand se doreste ca o interogare sa regaseasca inregistrari dintr-o tabela care indeplinesc o conditie ce depinde la randul ei de valori din aceeasi tabela.

**Nota:** Clauza ORDER BY nu poate fi utilizata intr-o subinterogare. Regula este ca poate exista doar o singura clauza ORDER BY pentru o comanda SELECT si, daca este specificata, trebuie sa fie ultima clauza din comanda SELECT. Prin urmare, clauza ORDER BY nu poate fi specificata decat in interogarea cea mai din exterior.

**Subinterogari care returneaza un singur rand**

In acest caz conditia, din clauza WHERE sau HAVING a interogarii exterioare utilizeaza operatorii: =, <, <=, >, >=, <> care opereaza asupra unei subinterogari ce returneaza o singura valoare. Interogarea interioara poate contine conditii complexe formate prin utilizarea conditiilor multiple de interogare cu ajutorul operatorilor AND si OR sau prin utilizarea functiilor agregat.

Urmatoarea interogare selecteaza cadrele didactice care au salariul cel mai mic. Salariul minim este determinat de o subinterogare ce returneaza o singura valoare.

SQL> SELECT nume, prenume, salariu

     FROM profesor

     WHERE salariu = (SELECT MIN (salariu)

                     FROM profesor);

NUME      PRENUME   SALARIU

----      -------   -------

VOINEA    MIRCEA    1200

STANESCU MARIA     1200

Procesul de evaluare al acestei interogari se desfasoara astfel:

        Se evalueaza in primul rand interogarea interioara:

        Valoarea obtinuta este MIN ( salariu ) = 1 200

        Rezultatul evaluarii interogarii interioare devine conditie de cautare pentru interogarea

        exterioara si anume:

SQL> SELECT nume, prenume, salariu

     FROM profesor

     WHERE salariu = 1200;

in cazul in care interogarea interioara nu intoarce nici o inregistrare, interogarea exterioara nu va selecta la randul ei nici o inregistrare.

**Nota:** Daca se utilizeaza operatorii: =, <, <=, >, >=, <> in conditia interogarii exterioare, atunci interogarea interioara trebuie in mod obligatoriu sa returneze o singura valoare. In caz contrar va aparea un mesaj de eroare, ca in exemplul urmator:

SQL> SELECT nume, prenume, salariu

     FROM profesor

     WHERE salariu = (SELECT MIN (salariu)

                     FROM profesor

                     GROUP BY grad);

ERROR:

ORA-01427: single-row subquery returns more than one row

Subinterogairile pot fi folosite nu numai in clauza WHERE ainterogarii exterioare, ci si in clauza HAVING. Urmatoarea interogare afiseaza toate gradele didactice pentru care salariul minim este mai mare decat salariul mediu al tuturor cadrelor didactice.

SQL> SELECT grad

FROM profesor

GROUP BY grad

HAVING MIN(salariu)>(SELECT AVG(salariu)

 FROM profesor);

GRAD

----

CONT

LECT

PROF

Subinterogari care returneaza mai multe randuri

In cazul cand interogarea intoarce mai multe randuri nu mai este posibila folosirea operatorilor de comparatie. In locul acestora se foloseste operatorul IN, care asteapta o lista de valori si nu doar una.

Urmatoarea interogare selecteaza pentru fiecare grad didactic acele persoane care au salariul minim. Salariul minim pentru fiecare grad didactic este aflat printr-o subinterogare, care, evident, va intoarce mai multe randuri:

SQL> SELECT nume, salariu, grad

     FROM profesor

     WHERE (salariu, grad) IN

                             (SELECT MIN (salariu), grad

                             FROM profesor

                             GROUP BY grad)

     ORDER BY salariu;

NUME      SALARIU   GRAD

----      -------   ----

VOINEA    1200      ASIST

STANESCU 1200      ASIST

ALBU      2200      LECT

MARIN     2500      PROF

GEORGESCU 2800     CONF

**Nota**: Spre deosebire de celelalte interogari de pana acum, interogarea de mai sus compara perechi de coloane. In acest caz trebuie respectate urmatoarele reguli:

        coloanele din dreapta conditiei de cautare sunt in paranteze si fiecare coloana este separata prin virgula;

        coloanele returnate de interogarea interioara trebuie sa se potriveasca ca numar si tip cu coloanele cu care sunt comparate in interogarea exterioara; in plus, ele trebuie sa fie in aceeasi ordine cu coloanele cu care sunt comparate.

Alaturi de operatorul IN, o subinterogare care returneaza mai multe randuri poate folosi operatorii ANY, ALL sau EXISTS. Operatorii ANY si ALL sunt prezentati in continuare, iar operatorul EXISTS va fi prezentat in sectiunea 'Subinterogari corelate'.

Operatorii ANY si ALL sunt folositi in mod obligatoriu in combinatie cu operatorii relationali =, ! =, <, >, <=, >=; operatorii IN si EXISTS nu pot fi folositi in combinatie cu operatorii relationali, dar pot fi utilizati cu operatorul NOT, pentru negarea expresiei.

Operatorul ANY

Operatorul ANY (sau sinonimul sau SOME) este folosit pentru a compara o valoare cu oricare dintre valorile returnate de o subinterogare. Pentru a intelege modul de folosire a acestui operator sa consideram urmatorul exemplu ce afiseaza cadrele didactice ce castiga mai mult decat profesorii care au cel mai mic salariu:

SQL> SELECT nume, salariu, grad

     FROM profesor

     WHERE salariu > ANY (SELECT DISTINCT salariu

                         FROM profesor

                         WHERE grad='PROF');

NUME           SALARIU   GRAD

----           -------   ----

GHEORGHIU     3000      PROF

GEORGESCU      2800      CONF

Interogarea de mai sus este evaluata astfel: daca salariul unui cadru didactic este mai mare decat cel putin unul din salariile returnate de interogarea interioara, acea inregistrare este inclusa in rezultat. Cu alte cuvinte, >ANY inseamna mai mare decat minimul dintre valorile returnate de interogarea interioara, <ANY inseamna mai mic ca maximul, iar =ANY este echivalent cu operatorul IN.

**Nota:** Optiunea DISTINCT este folosita frecvent atunci cand se foloseste operatorul ANY pentru a preveni selectarea de mai multe ori a unor inregistrari.

Operatorul ALL

Operatorul ALL este folosit pentru a compara o valoare cu *toate*valorile returnate de o subinterogare. Consideram urmatorul exemplu ce afiseaza cadrele didactice care castiga mai mult decat asistentii cu salariul cel mai mare:

SQL> SELECT nume, salariu, grad

     FROM profesor

     WHERE salariu > ALL (SELECT DISTINCT salariu

                         FROM profesor

                         WHERE grad='ASIST');

NUME      SALARIU   GRAD

----      -------   ----

GHEORGHIU 3000      PROF

MARIN     2500      PROF

GEORGESCU 2800      CONF

ALBU      2200      LECT

Interogarea de mai sus este evaluata astfel: daca salariul unui cadru didactic este mai mare decat toate valorile returnate de interogarea interioara, acea inregistrare este inclusa in rezultat. Cu alte cuvinte, >ALL inseamna mai mare ca maximul dintre valorile returnate de interogarea interioara iar <ALL inseamna mai mic ca minimul dintre acestea.

**Nota:** Operatorul ALL nu poate fi utilizat cu operatorul = deoarece interogarea nu va intoarce nici un rezultat cu exceptia cazului in care toate valorile sunt egale, situatie care nu ar avea sens.

**Subinterogari imbricate**

Subinterogarile pot fi imbricate (utilizate cu alte subinterogari) pana la 255 de nivele, indiferent de numarul de valori returnate de fiecare subinterogare. Pentru a selecta cadrele didactice care au salariul mai mare decat cel mai mare salariu al cadrelor didactice care apartin catedrei de Electronica, vom folosi urmatoarea interogare:

SQL> SELECT nume, prenume, salariu

     FROM profesor

     WHERE salariu >(SELECT MAX(salariu)

                   FROM profesor

                   WHERE cod\_catedra=(SELECT cod\_catedra

                                    FROM catedra

                                    WHERE nume= 'ELECTRONICA'));

Subinterogari corelate

In exemplele considerate pana acum interogarea interioara era evaluata prima, dupa care valoarea sau valorile rezultate erau utilizate de catre interogarea exterioara. Subinterogarile de acest tip sunt numite subinterogari simple. O alta forma de subinterogare o reprezinta *interogarea corelata,*caz in care interogarea exterioara transmite repetat cate o inregistrare pentru interogarea interioara. Interogarea interioara este evaluata de fiecare data cand este transmisa o inregistrare din interogarea exterioara, care se mai numeste si *inregistrare candidata.*Subinterogarea corelata poate fi identificata prin faptul ca interogarea interioara nu se poate executa independent ci depinde de valoarea transmisa de catre interogarea exterioara. Daca ambele interogari acceseaza aceeasi tabela, trebuie asigurate alias-uri pentru fiecare referire la tabela respectiva

Subinterogarile corelate reprezinta o cale de a accesa fiecare inregistrare din tabel si de a compara anumite valori ale acesteia cu valori ce depind tot de ea.

Evaluarea unei subinterogari corelate se executa in urmatorii pasi:

1.  Interogarea exterioara trimite o inregistrare candidata catre interogarea interioara;

2.  Interogarea interioara se executa in functie de valorile inregistrarii candidate;

3.  Valorile rezultate din interogarea interioara sunt utilizate pentru a determina daca inregistrarea candidata va fi sau nu inclusa in rezultat;

4.  Se repeta procedeul incepand cu pasul l pana cand nu mai exista inregistrari candidate.

De exemplu pentru a regasi cadrele didactice care castiga mai mult decat salariul mediu din propria catedra, putem folosi urmatoarea interogare corelata:

SQL> SELECT nume, prenume, salariu

     FROM profesor p

     WHERE salariu>(SELECT AVG(salariu)

                   FROM profesor s

                   WHERE s.cod\_catedra = p. cod\_catedra);

NUME      PRENUME   SALARIU

----      -------   -------

GHEORGHIU STEFAN    3000

MARIN     VLAD      2500

ALBU      GHEORGHE  2200

In exemplul de mai sus coloana interogarii exterioare care se foloseste in interogarea interioara este*p. cod\_catedra*. Deoarece *p. cod\_catedra* poate avea o valoare diferita pentru fiecare inregistrare, interogarea interioara se executa pentru fiecare inregistrare candidata transmisa de interogarea exterioara.

Atunci cand folosim subinterogari corelate impreuna cu clauza HAVING, coloanele utilizate in aceasta clauza trebuie sa se regaseasca in clauza GROUP BY. In caz contrar, va fi generat un mesaj de eroare datorat faptului ca nu se pate face comparatie decat cu o expresie de grup. De exemplu, urmatoarea interogare este corecta, ea selectand gradele didactice pentru care media salariului este mai mare decat maximul primei pentru acelasi grad:

SQL> SELECT grad

     FROM profesor p

     GROUP BY grad

     HAVING AVG (salariu)>(SELECT MAX(prima)

                          FROM profesor

                          WHERE grad = p.grad);

 grad

------

 ASIST

 CONF

Operatorul EXISTS

Operatorul EXISTS verifica daca, pentru fiecare inregistrare transmisa de interogarea exterioara, exista sau nu inregistrari care satisfac conditia interogarii interioare, returnand interogarii exterioare valoarea *True* sau *False*. Cu alte cuvinte, operatorul EXISTS cere in mod obligatoriu corelarea interogarii interioare cu interogarea exterioara. Datorita faptului ca operatorul EXISTS verifica doar existenta randurilor selectate si nu ia in considerare numarul sau valorile atributelor selectate, in subinterogare poate fi specificat orice numar de atribute; in particular, poate fi folosita o constanta si chiar simbolul \* (desi acest lucru nu este recomandabil din punct de vedere al eficientei). De altfel, EXISTS este singurul operator care permite acest lucru.

Urmatoarea interogare selecteaza toate cadrele didactice care au macar un subordonat:

SQL> SELECT cod, nume, prenume, grad

     FROM profesor p

     WHERE EXISTS

                (SELECT '1'

                 FROM profesor

                 WHERE profesor.sef = p.cod)

     ORDER BY cod;

--------------------------------------

 cod    nume      prenume        grad

--------------------------------------

 100  GHEORGHIU   STEFAN         PROF

 102  GEORGESCU   CRISTIANA      CONF

 103  IONESCU     VERONICA       ASIST

--------------------------------------

La fel ca si operatorul IN, operatorul EXISTS poate fi negat, luand forma NOT EXISTS. Totusi, o remarca foarte importanta este faptul ca pentru subinterogari, NOT IN nu este la fel de eficient ca NOT EXISTS. Astfel daca in lista de valori transmisa operatorului NOT IN exista una sau mai multe valori Null, atunci conditia va lua valoarea de adevar *False*, indiferent de celelalte valori din lista.

De exemplu, urmatoarea interogare incearca sa returneze toate cadrele didactice care nu au nici un subaltern**:**

SQL> SELECT nume, grad

     FROM profesor

     WHERE cod NOT IN SELECT sef

                     FROM profesor);

Aceasta interogari nu va intoarce nici o inregistrare deoarece coloana sef contine si valoarea Null. Pentru a obtine rezultatul corect trebuie sa folosim urmatoarea interogare:

SQL> SELECT nume, grad

     FROM profesor p

     WHERE NOT EXISTS (SELECT '1'

                      FROM profesor

                      WHERE sef=p.cod);

+----------+--------+

| nume     | grad   |

+----------+--------+

| MARIN    | PROF   |

| ALBU     | LECT   |

| VOINEA   | ASIST  |

| STANESCU | ASIST  |

+----------+--------+

In general, operatorul EXISTS se foloseste in cazul subinterogarilor corelate si este cateodata cel mai eficient mod de a realiza anumite interogari. Performanta interogarilor depinde de folosirea indecsilor, de numarul randurilor returnate, de dimensiunea tabelei si de necesitatea crearii tabelelor temporare pentru evaluarea rezultatelor intermediare. Tabelele temporare generate de Oracle nu sunt indexate, iar acest lucru poate degrada performanta subinterogarilor daca se folosesc operatorii  IN,  ANY sau ALL.

Subinterogarile mai pot aparea si in alte comenzi SQL cum ar fi: UPDATE, DELETE, 1NSERT si CREATE TABLE.

Asa cum am vazut, exista in principal doua moduri de realizare a interogarilor ce folosesc date din mai multe tabele: jonctiuni si subinterogari. Jonctiunile reprezinta forma de interogare *relationala* (in care sarcina gasirii drumului de acces la informatie revine SGRD-ului) iar subinterogarile forma procedurala (in care trebuie indicat drumul de acces la informatie). Fiecare dintre aceste forme are avantajele sale, depinzand de cazul specific in care se aplica.

**5.1.21. Operatii pe tabele ce contin informatii do structura arborescenta**

O baza de date relationala nu poate stoca inregistrari in mod ierarhic, dar la nivelul inregistrarii pot exista informatii care determina o relatie ierarhica intre inregistrari. SQL permite afisarea randurilor dintr-o tabela tinand cont de relatiile ierarhice care apar intre randurile tabelei. Parcurgerea in mod ierarhic a informatiilor se poate face doar la nivelul unei singure tabele. Operatia se realizeaza cu ajutorul clauzelor START WITH si CONNECT BY din comanda SELECT.

De exemplu, in tabela profesor exista o relatie ierarhica intre inregistrari datorata valorilor din coloanele *cod* si *sef*. Fiecare inregistrare aferenta unui cadru didactic contine in coloana *sef* codul persoanei careia ii este direct subordonat. Pentru a obtine o situatie ce contine nivelele ierarhice, vom folosi urmatoarea interogare:

SQL> SELECT LEVEL, nume, prenume, grad

     FROM profesor

     CONNECT BY PRIOR cod=sef

     START WITH sef IS NULL;

LEVEL     NUME           PRENUME        GRAD

------    -----          -------        ----

1         GHEORGHIU      STEFAN         PROF

2         MARIN        VLAD          PROF

2         GEORGESCU    CRISTIANA     CONF

3         IONESCU      VERONICA     ASIST

4         STANESCU     MARIA        ASIST

2         ALBU           GHEORGHE     LECT

2         VOINEA        MIRCEA       ASIST

Explicarea sintaxei si a regulilor de functionare pentru exemplul de mai sus:

        Clauza standard SELECT poate contine pseudo-coloana LEVEL ce indica nivelul inregistrarii in arbore (cat de departe este de nodul radacina). Astfel, nodul radacina are nivelul l, fiii acestuia au nivelul 2, s.a.m.d.;

        In clauza FROM nu se poate specifica decat o tabela;

        Clauza WHERE poate aparea in interogare pentru a restrictiona vizitarea nodurilor  (inregistrarilor) din cadrul arborelui;

        Clauza CONNECT  BY specifica coloanele prin care se realizeaza relatia ierarhica; acesta este clauza cea mai importanta pentru parcurgerea arborelui si este obligatorie;

        Operatorul PRIOR stabileste directia in care este parcurs arborele. Daca clauza apare inainte de atributul cod, arborele este parcurs de sus in jos, iar daca apare inainte de atributul s*ef* arborele este parcurs de jos in sus;

        Clauza START WITH specifica nodul (inregistrarea) de inceput a arborelui. Ca punct de start nu se poate specifica un anumit nivel (LEVEL), ci trebuie specificata valoarea; aceasta clauza este optionala, daca ea lipseste, pentru fiecare inregistrare se va parcurge arborele care are ca radacina aceasta inregistrare.

In sintaxa interogarii de mai sus, pentru a ordona inregistrarile returnate, poate aparea clauza OROER BY, dar este recomandabil sa nu o folosim deoarece ordinea implicita de parcurgere a arborelui va fi distrusa.

Pentru a elimina doar un anumit nod din arbore putem folosi clauza WHERE, iar pentru a elimina o intreaga ramura dintr-un arbore (o anumita inregistrare impreuna cu *fiii* acesteia) folosim o conditie compusa in clauza CONNECT  BY.

Urmatorul exemplu elimina doar inregistrarea cu numele 'GEORGESCU', dar nu si *fiii*acesteia:

SQL> SELECT LEVEL, nume, prenume, grad

     FROM profesor

     WHERE nume != 'GEORGESCU'

     CONNECT BY PRIOR cod=sef

     START WITH sef IS NULL;

LEVEL NUME         PRENUME   GRAD

----- ----         -------   ----

1     GHEORGHIU    STEFAN    PROF

2     MARIN        VLAD      PROF

3     IONESCU     VERONICA  ASIST

4     STANESCU     MARIA     ASIST

2     ALBU         GHEORGHE  LECT

2     VOINEA       MIRCEA    ASIST

Pentru a elimina toata ramura care contine inregistrarea cu numele 'GEORGESCU' si inregistrarile pentru subordonatii acesteia se foloseste urmatoarea interogare:

SQL> SELECT LEVEL, nume, prenume, grad

     FROM profesor

     CONNECT BY PRIOR cod=sef AND nume != 'GEORGESCU'

     START WITH sef IS NULL;

LEVEL     NUME      PRENUME   GRAD

-----     ----      -------   ----

1         GHEORGHIU STEFAN    PROF

2         MARIN     VLAD      PROF

2         ALBU      GHEORGHE  LECT

2         VOINEA    MIRCEA    ASIST